JET: Dynamic Join-Exit-Tree Amortization and Scheduling for Contributory Key Management

Yinian Mao*, Yan Sun†, Min Wu* and K. J. Ray Liu*

* Department of Electrical and Computer Engineering, University of Maryland, College Park

† Department of Electrical and Computer Engineering, University of Rhode Island

Abstract

In secure group communications, the time cost associated with key updates in the events of member join and departure is an important aspect of quality of service, especially in large groups with highly dynamic membership. To achieve better time efficiency, we propose a join-exit-tree (JET) key management framework. First, a special key tree topology with join and exit subtrees is introduced to handle key updates for dynamic membership. Then optimization techniques are employed to determine the capacities of join and exit subtrees for achieving the best time efficiency, and algorithms are designed to dynamically update the join and exit trees. We show that on average, the asymptotic time cost for each member join/departure event is reduced to $O(\log(\log n))$ from the previous cost of $O(\log n)$, where n is the group size. Our experimental results based on simulated user activities as well as the real MBone data demonstrate that the proposed JET scheme can significantly improve the time efficiency, while maintaining low communication and computation cost, of tree-based contributory key management.

Index Terms

secure group communications, time efficiency, contributory key management, dynamic tree topology.

Part of this work was presented in IEEE INFOCOM'04, Hong Kong, China. This work was supported in part by the Army Research Office under Award No. DAAD19-01-1-0494 and the National Science Foundation under Award No. CCR-0133704. Corresponding author's Email: ymao@eng.umd.edu, Tel 301-405-3347, Fax 301-314-9281.

I. Introduction

The advances in communication and networking technologies have paved ways for people to share and disseminate information. Along with the growing exchange of information, the security of communications has drawn increasing attention. An important aspect of communication security is content confidentiality and access control [1], which becomes a necessity in a wide range of applications, such as bank transactions, teleconferencing, and data collection in sensor networks [2] [3]. For secure group-oriented applications, access control is a challenging task due to the potentially large group size and dynamic membership.

To achieve confidentiality in group communications, a key known to all group members is used to encrypt the communication content [4] [5]. This key is usually referred to as the *group key* [6]–[8]. In a group with dynamic membership, the group key needs to be updated upon each user's join to prevent the new user from accessing the past communications. Similarly, upon each user's departure, the key needs to be updated to prevent the leaving user from accessing the future communications. Thus group members need to agree upon the same key management protocol for key establishment and update. Sometimes the group key management protocol is also referred to as the *group key agreement*.

A group key management scheme follows either a centralized or a contributory approach. The centralized approach uses a central key server to generate and distribute keys for all group members [6] [9] [10], whereas in the contributory approach, each group member contributes his/her own share to the group key [11]–[13]. Since contributory schemes do not rely on a central key server, they become necessary in situations where: (a) a central key server cannot be established, such as in Ad Hoc networks, (b) group members do not trust another entity to manage their private keys, or (c) members and server do not share any common knowledge about each other's secret keys beforehand. Contributory schemes remove the need of the key server at the expense of performing computationally expensive cryptographic primitives, such as modular multiplication and exponentiation [4] [14], posing a challenge to the design of efficient contributory key agreements.

In the literature, many group key management protocols have been proposed [6], [7], [9]–[13], [15]–[24]. The early designs of contributory key agreements mostly consider the efficiency of key establishment [25] [26] [27]. Among them, Ingemarsson *et al.* first introduced a conference key distribution system (CKDS) based on a ring topology [25]. Later, Burmester and Desmedt proposed a key distribution system (BD) that takes only three rounds to generate a group key [28]. Steiner *et al.* extended the two-party Diffie-Hellman (DH) protocol and proposed group Diffie-Hellman protocols GDH.1/2/3 [26].

Becker and Willie studied the minimum communication complexity of contributory key agreements and proposed the *octopus* and 2^d*-octopus* protocols [27], which have proven optimality for key establishment. While achieving efficiency in key establishment, most of these early schemes encounter high rekeying complexity in either member join or departure. Recent research on key management became more aware of the scalability issue. As a means to improve scalability, tree-based approach for group rekeying was first presented in the centralized scenario by Wallner *et al.* in [9] and Wong *et al.* in [6], independently. Later, tree-based schemes were also proposed for the contributory setting by Kim *et al.* in their TGDH scheme [12], and by Dondeti *et al.* in their DISEC scheme [13]. The tree-based schemes use a logical *key tree* to organize the keys belonging to the group members and achieve a rekeying complexity of $O(\log n)$ [6] [12] [13] [21], where n is the group size. In addition, [12] and [13] also pointed out that the rekeying cost is related to both the key tree structure and the location of member join or departure in the key tree, and suggested a balanced key tree to reduce the rekeying cost based on heuristics. In [23], Zhu *et al.* proposed two schemes to optimize the rekeying cost in centralized key management. The key tree structure is re-organized according to the temporal patterns of the group members, or the packet loss probability along the route from the key server to each member.

In this paper, we investigate the time efficiency of contributory key agreement. The time efficiency is measured by the processing time in group key establishment and update. In order to participate in the group communications, a joining user has to wait until the group keys are updated. Since computing cryptographic primitives and exchanging rekeying messages are time-consuming, such waiting time is not negligible. Similarly, the amount of time needed to recompute a new group key reflects the latency in user revocation. Thus from a quality of service (QoS) perspective, the rekeying time cost is directly related to users' satisfaction and a system's performance. Traditionally, the rekeying time complexity is analyzed only for one join or departure event. The design rationale of our scheme is to look into the combination of multiple events, and optimize the time cost over the dynamics of group membership. To improve the time efficiency, we design a new key tree topology with join and exit subtrees, which are small subtrees located close to the root of the key tree. With this key tree topology, we propose a set of algorithms to handle the key update for join and leave events. In particular, we show through analysis that the sizes of join and exit trees should be at the log scale of the group size. The resulting scheme is called *Join-Exit Tree (JET) Group Key Agreement*. Analytical results show that the proposed scheme achieves an average asymptotic time cost of $O(\log(\log n))$ for a join event, and also $O(\log(\log n))$ for a departure event when group dynamics are known *a priori*. In addition to the improved time efficiency, our scheme also has low communication and computation complexity.

The rest of this paper is organized as follows. Section II discusses the efficiency issues in contributory key agreements and proposes a few performance metrics. Section III presents the join and exit tree topology and algorithms in our scheme. These algorithms are integrated into a unified protocol in Section IV. We present the simulation results in Section V, then discuss protocol implementation and other efficiency aspects in Section VI. Finally, the conclusions are drawn in Section VII.

II. Efficiency Aspects in Contributory Key Agreement

A. Background on Tree-based Contributory Key Management

We briefly review rekeying operations for join and leave events in tree-based contributory key agreements [13] [12], which use the two-party DH protocol [29] as a basic module.

In a tree-based key agreement, three types of keys are organized in a logical key tree, as illustrated in Fig. 1(a). The leaf nodes in a key tree represent the private keys held by individual group members. The root of the tree corresponds to the group key. All other inner nodes represent subgroup keys, each of which is held by the group members that are descendants of the corresponding inner node. We denote the i-th group member by M_i, and the key associated with the j-th node in the key tree by K_j. In addition, g and p are the exponentiation base and the modular base for the DH protocol, respectively.

To establish a group key, the keys in the key tree are computed in a bottom-up fashion. Users are first grouped into pairs and each pair performs a two-party DH to form a sub-group. These sub-groups will again pair up and perform the two-party DH to form larger sub-groups. Continuing in this way, the final group key can be obtained. An example is shown in Fig.1(a) with four group members, and member M_i has private key r_i. The group key K_1 corresponding to node 1 is computed in two rounds as

$$K_1 = g^{(g^{r_1 r_2} \bmod p)(g^{r_3 r_4} \bmod p)} \bmod p.$$

In a user join event, the new user will first pair up with an insertion node, which could be either a leaf node or an inner node, to perform a two-party DH. Then all the keys on the path from the insertion node to the tree root are updated recursively. An example is shown in Fig.1. When member M_5 joins the group, node 7 in Fig.1(a) is chosen as the insertion node. Then M_4 (node 7) and M_5 (node 9) perform a DH key exchange to generate a new inner node 8 in Fig. 1(b), followed by the key updates on the path $node\ 8 \rightarrow node\ 3 \rightarrow node\ 1$.

Upon a user's departure, the leaving user's node and its parent node will be deleted from the key tree. Its sibling node will assume the position of its parent node. Then all the keys on the path from the leaving user's grandparent node to the tree root are updated from the bottom to the top.

B. Time-Efficiency Issues in Contributory Key Agreements

The time efficiency of DH-based contributory group key agreement is usually evaluated by the number of rounds needed to perform the protocol during a key update [12], [13], [25], [26]. However, in some schemes, the number of operations may be different in distinct rounds. For example, in GDH.2 [26], i modular exponentiations are performed in the i-th round. To address this problem, the notion of "simple round" was introduced in [27], where every party can send and receive at most one message in each round. In our work, we apply the notion of simple round in the tree-based contributory schemes. In each round, each user can perform at most one two-party DH operation. With the new definition of round, we propose performance metrics for time efficiency below.

Average Join/Leave Time We define the *user join time* as the number of rounds to process key updates for a user join event. The average user join time, denoted by T_{join}, is defined as

$$T_{join} = \frac{R_{join}}{N_{join}}, \tag{1}$$

where R_{join} is the total number of DH rounds performed for N_{join} join events. Similarly, the *user leave time* is defined as the number of rounds to process key updates for a user leave event. The average user leave time, denoted by T_{leave}, is defined as

$$T_{leave} = \frac{R_{leave}}{N_{leave}}, \tag{2}$$

where R_{leave} is the total number of DH rounds performed for N_{leave} leave events. Let $N = N_{join} + N_{leave}$ and $R = R_{join} + R_{leave}$. The overall average processing time T is defined as

$$T = \frac{R}{N}, \tag{3}$$

where T can also be interpreted as a weighted average of T_{join} and T_{leave} as $T = \frac{N_{join}}{N} T_{join} + \frac{N_{leave}}{N} T_{leave}$.

C. Communication and Computation Efficiency

The communication efficiency of a contributory key agreement refers to the number of messages sent for a key update during a join or leave event. The underlying assumption is that sending each message incurs about the same communication cost. In practice, the size of each message could be different. However, the main cost in sending a rekeying message is the cost in software and hardware to go through the protocol stack and form a packet, along with the cost in networks while routing and transmitting the packet. Similar to the case of time efficiency, we choose the average number of messages as the performance metric for communication efficiency.

In a DH-based contributory group key agreement, the computation of modular exponentiation dominates the total computation cost. We use the average number of exponentiations for join and departure events as the performance metrics for the computation efficiency.

III. Join-Exit Tree (JET) Algorithms

In this section, we present a new logical key tree topology and the associated algorithms to achieve better time efficiency in contributory key agreement. As shown in Fig. 2(a), the proposed logical key tree consists of three parts: the *join tree*, the *exit tree*, and the *main tree*. The proposed key tree is a binary tree built upon the two-party DH protocol. We refer to the key tree in Fig. 2(a) as a *join-exit tree* and a key tree without special structures as a *simple key tree*. The prior works have shown that, if a user joins the group at a location closer to the tree root, fewer number of keys need to be updated, thus the join time will be shorter. Similar reasoning applies to user departures. So the join tree and exit trees should be much smaller than the main tree. We define the *join tree capacity* and the *exit tree capacity*, denoted by C_J and C_E, as the maximum number of users that can be accommodated in the join and exit tree, respectively. The number of users in the join tree and the main tree are denoted by N_J and N_M, respectively.

A joining user will first be added to the join tree. Later on, when the join tree reaches its capacity, all users in the join tree will be relocated together into the main tree. In addition, when users' departure time is known, users who are most likely to leave in the near future will be moved in batch from the main tree to the exit tree. The design rationale of the join and exit trees resembles that of memory hierarchy in computer design [30]. Furthermore, the capacities of the join and exit trees can change over time, resulting in a dynamic key tree structure. For example, when there is no user in the exit tree, the key tree reduces to a main tree and join tree topology, as shown in Fig. 2(b).

A. The Join Tree Algorithm

The join tree algorithm consists of four parts: the join tree activation, the insertion strategy, the relocation strategy, and the join tree capacity update. When the group has only a few members, the join tree is not activated. As the group size increases and exceeds a threshold we activate the join tree and choose an initial join tree capacity. Such a threshold condition is referred to as the *activation condition* for the join tree. After the activation, any user joining the group is first inserted to a node in the join tree. The insertion node is chosen according to the *insertion strategy*. When the join tree is full, the members in the join tree are merged into the leaf nodes of the main tree. Such a process is called the *batch relocation*. Since the number of users in the main tree is changed after the batch relocation, the join tree capacity is updated according to a rule that relates the join tree capacity to the main tree user number. According to this rule, the *optimal join tree capacity* in the sense of time efficiency can be computed. We explain these four parts in details below.

1) User Insertion in the Join Tree: When the join tree is empty and a new user wants to join, the root of the current key tree is chosen as the insertion node. The insertion is done by treating the entire existing group as one logical user, and performing a two-party DH between this logical user and the new user. This process is illustrated in Fig. 3, where the new user M_5 becomes node 9, the root of the join tree. Member M_5 is paired up with the original root of the key tree (node 1) to perform a DH key exchange and the new group key is established as node 8. When the join tree is not empty, the insertion node is determined by Algorithm 1, where $usernumber(x)$ returns the number of users under a given node x in the key tree. After the insertion node is found, the new member node performs a two-party DH key exchange with the insertion node. Then the keys on the path from the insertion node to the tree root are updated through a series of DH key exchange. Fig.4 illustrates the growth of the join tree from 1 user to 8 users using the insertion strategy.

Algorithm 1 Finding the insertion node

```
$x \leftarrow$ *join-tree-root*
**while** $usernumber(x) \neq 2^k$ for some integer $k$ **do**
  $x \leftarrow rightchild(x)$
**end while**
*insertion-node* $\leftarrow x$
```

2) The Batch Relocation: We present two relocation methods that differ in whether the subgroup keys in the join tree are preserved. In the first method, all users in the join tree are viewed as a logical user during relocation, and this logical user is inserted into the shortest-depth leaf node of the main tree. Thus, the subgroup keys among the users in the join tree are preserved. This process is shown in Fig.5(a). Then all keys along the path from the insertion node to the tree root are updated, which is indicated by the dash line in Fig. 5(a). The reason to choose the shortest branch leaf node in the main tree as the insertion node is to guarantee that the relocation time is at most the log of the main tree size ($\lceil \log N_M \rceil$), because the

TABLE I

LATENCY OF SEQUENTIAL USER JOIN

k	1	2	3	4	5	6	7	8	9	10	...
$r(k)$	1	2	2	3	2	3	3	4	2	3	...

shortest branch must be smaller or equal to the average length of the branches, which is $\lceil \log N_M \rceil$ [1]. The only exception comes when the main tree is a complete balanced tree, the relocation time is $\log N_M + 1$, because one more level of the key tree must be created to accommodate the new logical user.

In the second relocation method, we find the N_J shortest-depth leaf nodes in the main tree as the insertion nodes for N_J join tree user. These insertion nodes are found so that the unbalance-ness of the key tree can be alleviated by the relocation process. Then we relocate the join tree users simultaneously to the insertion nodes. The keys on the branches from all original join tree users to the tree root are updated in parallel and finally a new group key is obtained. This process is illustrated in Fig.5(b). To analyze the time complexity, we note that this relocation may fill up the empty nodes at the shortest-depth leaf nodes of the main tree. The maximum depth of any relocation path would not exceed $\lceil \log(N_M + N_J) \rceil$. Since the join tree is much smaller than the main tree, the relocation time is upper bounded by $\lceil \log N_M \rceil + 1$.

Although the two relocation methods have similar time complexity, the first method will generally produce a skewed main tree. Since users may leave from a branch longer than the average depth of the key tree, an unbalanced key tree may cause the user departure time to be longer than the case when a balanced key tree is used. The second relocation method helps maintain the balance of the key tree, which reduces the expected cost of leave events [12]. We shall choose the second relocation method in this work because it takes into consideration both the join and leave time cost.

3) The Optimal Join Tree Capacity: Using the proposed insertion strategy, the user join latency for the k-th user in the join tree is measured as $r(k)$ rounds, which is listed in Table I. We observe a special property of the sequence $r(k)$, namely,

$$r(2^p + q) = 1 + r(q), \qquad 0 < q \leq 2^p, \tag{4}$$

where p is a non-negative integer, and q a positive integer. For the user join latency $r(k)$ in (4), the following inequality holds for any positive integer n, and equality is achieved when n is of power of 2:

[1]Throughout this paper, $\log$ stands for base-2 logarithm and $\ln$ stands for natural logarithm.

$$\frac{1}{n}\sum_{k=1}^{n} r(k) \leq \frac{1}{2}\log n + 1. \tag{5}$$

The proof is presented in Appendix I.

Consider the average join time for x users joining the group starting form an empty join tree. These x users are inserted into the join tree one by one, then they are relocated together into the main tree. From previous analysis we can see that, when the main tree has N_M users, the average join tree relocation time is $\log N_M$, where we relax the integer value of the tree height to a continuous value to simplify analysis. Taking into account the relocation time, the average join time for these x users is

$$T_{join} = \frac{1}{x}(\sum_{k=1}^{x} r(k) + \log N_M). \tag{6}$$

Using (5), we obtain

$$T_{join} \leq \frac{1}{2}\log x + \frac{1}{x}\log N_M + 1. \tag{7}$$

Since it is not easy to minimize T_{join} directly, we try to minimize its upper bound over x. The optimal join tree capacity C_J that minimizes the upper bound is given by

$$\begin{aligned} C_J &= \text{arg min}_{x>0}\{\frac{1}{2}\log x + \frac{1}{x}\log N_M + 1\} \\ &= 2\ln N_M \end{aligned} \tag{8}$$

The above analysis shows that, for a given number of main tree users N_M and the insertion rule specified by Algorithm 1, the optimal join tree capacity C_J is $2\ln N_M$. Since between two consecutive join tree relocations, the main tree size is fixed at N_M, the join tree capacity should also be fixed during this time at $C_J \approx 2\ln N_M$ and the average join time is upper bounded by

$$T_{join} \leq \frac{1}{2}\log\log N_M + \frac{3}{2} + \frac{1}{2\ln 2} - \frac{1}{2}\log\log e. \tag{9}$$

This upper bound indicates that on average, a user needs to spend only $O(\log\log n)$ rounds for a rekeying operation in user join, where n is the group size. We note that this asymptotic performance is not affected by the variation of the relocation time, because the relocation time of around $\log N_M$ rounds is averaged over $\log N_M$ join events, contributing approximately only one round to the average join cost. This validates the use of the approximate average relocation time $\log N_M$ in the above analysis.

For the joining users, since they can start to communicate once they are inserted into the join tree, their waiting time do not include the relocation time of $\log N_M$ rounds. We refer to the waiting time for the joining users as *user join latency*. We can see that the average user join latency, L_{join}, is also upper bounded as

$$L_{join} \leq \frac{1}{2}\log(\log N_M) - \frac{1}{2}\log\log e + \frac{3}{2}.$$

4) The Join Tree Activation: To decide whether to activate the join tree, we compare the average join time with and without employing the join tree. For a key tree structure with join tree, adding each user in the join tree incurs at most a time cost of $\log C_J$ rounds. Consider the average user join time for C_J users when the join tree changes from empty to full, followed by a batch relocation of $\log N_M$ rounds. The average join time for these C_J users satisfies

$$T_{join} \leq \log C_J + (\log N_M)/C_J. \tag{10}$$

If a simple key tree with only a main tree is used, the average join time would be at least $\log N_M$. Consequently, a reduction in time cost can be obtained by using the join tree when the following inequality holds,

$$\log C_J + (\log N_M)/C_J \leq \log N_M,$$

or equivalently,

$$\log N_M \geq \frac{C_J}{C_J - 1}\log C_J. \tag{11}$$

We can see that when the number of users in the group is large enough, a join tree should be activated to reduce the average join time. In Appendix II, we show that when $C_J = 2\ln N_M$, the inequality (11) is satisfied for any $N_M > 8$. Thus we have found a threshold group size $TH_{join} = 8$. When the group size is smaller than or equal to 8, a simple key tree is used. Otherwise, the join tree is activated.

B. The Exit Tree Algorithm

In some group applications, users can estimate the duration of their staying time according to their own schedule. Such information can help reduce the time cost of rekeying operations in user departure. In the following analysis, we assume that we can obtain accurate information about users' duration of stay. In later sections, the cases of inaccurate or unavailable staying time will be discussed.

Similar to the join tree algorithm, the exit tree algorithm consists of four parts, namely, the activation condition, the batch movement, the user insertion in the exit tree, and the optimization of the exit tree capacity.

1) The Batch Movement: The *batch movement* refers to the operations to move the users that are likely to leave in the near future from the main tree to the exit tree. The group communications is not interrupted since the old group key can still be used before the batch movement is completed.

A batch movement takes place when there is a user leaving from the exit tree and a batch movement condition is satisfied. Denoting the number of users in the exit tree after the last batch movement as U_p, and the current number of users in the exit tree as U_c, we propose a batch movement condition as

$$U_c \leq \rho U_p, \tag{12}$$

where $\rho \in [0, 1)$ is the *exit tree residual rate* (residual rate for short), a pre-determined parameter to control the timing of batch movement. In a batch movement, the first B users who are most likely to leave soon are moved to the exit tree, where B is referred to as the *batch movement size*. Starting from an empty exit tree ($U_p = 0$), the number of users in the exit tree after the k-th batch movement will be $\sum_{i=0}^{k-1} \rho^i B$. As k goes to infinity, the number of users in the exit tree converges to $B/(1-\rho)$. Therefore the exit tree capacity C_E is related to the batch movement size by

$$C_E = B/(1-\rho). \tag{13}$$

We propose to use a priority queue [31] to keep the departure time of all the users in the main tree. This queue is referred to as the *leaving queue*. The users' departure time is obtained from their arrival time and their estimated staying time. The leaving queue will be update under two circumstances. First, after a batch relocation of the join tree, the departure information of the join tree users are added to the leaving queue. Second, after the batch movement of the exit tree, the departure information of the moved users are removed from the leaving queue.

2) User Insertion in the Exit Tree: The insertion locations for the users being moved into the exit tree are chosen to maintain the balance of the exit tree. For each user insertion, the leaf node with the minimum depth in the exit tree is chosen as the insertion node.

3) Optimal Exit Tree Capacity: Here we derive the optimal exit tree capacity that minimizes an upper bound of the average leaving time. Suppose that b users are moved together into the exit tree. A batch movement of these b users will incur a time cost of $(\log N_M + 2)$, where $\log N_M$ is the average height of the main tree, and the addition of 2 refers to the additional two levels above the main tree due to the use of the join tree and the exit tree (refer to Fig. 2(a)). If the exit tree capacity is x, each user leaving from the exit tree will incur at most a time cost of $(\log x + 2)$. Thus the average user leave time for these b users is bounded by

$$T_{leave} \leq \frac{1}{b}(\log N_M + 2) + (\log x + 2). \tag{14}$$

Using (13), $b = x(1-\rho)$, and minimizing the right hand side of (14) we obtain

$$\begin{aligned} C_E &= \text{arg min}_x \left\{ \frac{1}{(1-\rho)x}(\log N_M + 2) + (\log x + 2) \right\} \\ &= \frac{\ln N_M + 2\ln 2}{(1-\rho)}. \end{aligned} \tag{15}$$

When the capacity of the exit tree is computed as in (15), the average leave time is bounded by

$$T_{leave} \leq \log(\log N_M + 2) + \delta, \tag{16}$$

where $\delta = 2 - \log(1-\rho) + \log e - \log\log e$. Combining (15) and (13), we have

$$B = \ln N_M + 2\ln 2. \tag{17}$$

A few comments should be made to provide more insights from the above analysis. First, the batch movement size B is only determined by the number of users in the main tree, and independent of the residue rate ρ. Second, there are actually only two parameters, B and ρ, in our system, since the exit tree capacity is a function of B and ρ as in (13). Third, with perfect departure information, the average leave time is bounded by $O(\log\log n)$, where n is the group size, and the residue rate ρ should be set to 0 to minimize the upper bound in (16). However, in practice, the choice of ρ is a tradeoff. When ρ is 0, a batch movement cannot be performed unless the exit tree is completely vacant. If some users inaccurately estimate their departure time and stay in the exit tree for a long period of time, no other users can utilize the exit tree during that period. When ρ is close to 1, batch movements are frequently performed, resulting in a large overhead. Based on experimental heuristics, we suggest setting ρ to around 0.5 .

4) The Activation of Exit Tree: The average leave time using a simple key tree with N_M users is $\log N_M$. Comparing this result with the upper bound in (14), a reduction in the average leave time can be obtained if

$$\frac{1}{(1-\rho)C_E}(\log N_M + 2) + (\log C_E + 2) \leq \log N_M. \tag{18}$$

Using (15), we simplify the above condition as

$$\log N_M \geq \log C_E + \log e + 2. \tag{19}$$

Similar to the case of the join tree activation, we can prove that when the exit tree capacity is chosen as in (15), the inequality (19) is satisfied for any $N_M > 256$. Thus we have found a threshold group size

$TH_{leave} = 256$. When the group size is larger than this threshold, activating the exit tree can reduce the average leave time.

IV. Group Key Agreement Based on Join-Exit Tree

In this section we present a protocol suite of the *Join-Exit Tree (JET) Group Key Agreement*, which consists of a key establishment protocol, a user join protocol, and a user leave protocol. These protocols are based on the algorithms we discussed in the previous section.

A. Group Key Establishment

Many prior works [21] [25] assume that all group members are available before starting the group communications, thus parallel computation can take place to establish a group key. We refer to this situation as *concurrent user join*. In reality, there are situations when members join the group sequentially, and we refer to them as *sequential user join*. The proposed JET scheme treats the key establishment in these two types of situations differently. For concurrent user join, subgroup keys in the key tree are computed in a bottom-up fashion in parallel to obtain the final group key, as in [21]. For sequential user join, we use the join protocol (as discussed below) to handle the sequential key updates. The join tree is activated when the group size exceeds the activation threshold $TH_{join} = 8$, but the exit tree will not be activated during the key establishment stage.

B. Join Protocol

The key update for a user join event follows the next few steps:

1. Choose an insertion node in the key tree.

(a) Before the join tree is activated, Algorithm 1 is used in the simple key tree to choose the insertion node.

(b) After the join tree is activated, when inserting the new user according to Algorithm 1 will not make the join tree height more than $\lceil \log C_J \rceil$, the insertion strategy in Algorithm 1 is followed. Otherwise, the insertion node will be chosen as the leaf node with the minimum depth in the join tree. (When there are user departures from the join tree, this helps keep the join tree balanced.)

2. The insertion node and the new member perform a two-party DH key exchange. Then all the keys on the path from the insertion node to the root are updated subsequently.

3. Adjust the key tree topology and parameters according to the rules specified as follows:

(a) When the group size is larger than $TH_{join} = 8$, the join tree is activated. When the group size is larger than $TH_{leave} = 256$, the exit tree is activated.

(b) When the join tree becomes full after a join event, users in the join tree are relocated into the main tree using the relocation strategy in Section III. Additionally, the departure information of those users who can report their staying time is stored in the leaving queue.

(c) Update the join and exit tree capacities according to Eqn. (8) and Eqn. (15), respectively.

C. Leave Protocol

The exit tree residual rate is set to $\rho = 0.5$. The key update for a user leave event follows the next few steps:

1. Delete the leaving user node and its parent node. Promote the leaving user's sibling node to their parent node's position. Mark the keys on the path from the leaving user's grandparent node to the tree root as to be updated later.

2. When the user is leaving from the main tree and there are also users in the join tree, perform a join tree relocation. Mark the keys to be updated for relocation.

3. Update all the keys marked in step 1 and 2 from the bottom to the top of the key tree.

4. When the user is leaving from the exit tree and the batch movement condition is satisfied, perform a batch movement as specified in Section III.

5. Perform the updates for key tree management as follows:

(a) Remove the leaving user's departure information if it is in the leaving queue.

(b) Compute the new join and exit tree capacities according to Eqn. (8) and Eqn. (15), respectively. If the newly-computed join/exit tree capacity becomes larger than the current number of users in the join/exit tree, the join/exit tree capacity is updated immediately. Otherwise, no update is done.

(c) When the main tree user number N_M falls below the threshold for join/exit tree activation and the join/exit tree is empty, the join/exit tree is disabled.

V. Experiments and Performance Analysis

In this section, we present three simulations. The first simulation focuses on group key establishment, in which we consider sequential user join. The second and third simulation have both join and departure activities. In each simulation, the performance of our proposed scheme is compared with that of TGDH scheme [12], a typical tree-based contributory key agreement.

A. Key Establishment for Sequential User Join

For sequential user join, the proposed JET protocol uses a simple key tree for small group size, and activates the join tree when the group size is larger than 8. The exit tree will not be activated. We compare the average join time for sequential user join using the proposed JET and TGDH [12] in Fig. 6. It can be seen that JET achieves the same performance as TGDH when the group size is small, and outperforms TGDH when the group size becomes large. Regarding the asymptotic performance, TGDH achieves an average time cost of $O(\log n)$, while the proposed JET scheme achieves $O(\log(\log n))$. The dashed line in Fig. 6 shows the theoretical upper bound for the average time cost from (9).

B. Experiment Using MBone User Activity Data

We choose three user activity log files from three Multicast Backbone (MBone) multicast sessions [32] as user activity for our simulation. Two of these three sessions are NASA space shuttle coverage and the other one is CBC News World online test [2].

Fig. 7 shows the experimental results using JET and TGDH scheme, where we can see that JET has about 50% improvement over TGDH in user join, and about 20% improvement in user departure. It is worth noting that the improvement in user departure is not resulted from the use of the exit tree, since all the three sessions have maximum group size below 100 and the exit tree is not activated. From the study of the MBone multicast sessions, Ammeroth *et al.* observed that the MBone multicast group size is usually small (typically 100-200), and users either stay in the group for a short period of time or a very long time [33] [34]. Using the proposed JET scheme, the exit tree will not be activated for a small group size. However, when a user stays in the group for only a short period of time, it is highly likely that this user joins and leaves the group in the join tree without getting to the main tree. Thus the use of the join tree reduces both the user join time and the user leave time.

C. Experiments Using Simulated User Activity Data

In this experiment, we generate user activities according to the probabilistic model suggested in [33]. The duration of simulation is 5000 time units and is divided into four non-overlapping segments, T_1 to T_4. In each time segment T_i, users' arrival time is modelled as a Poisson process with mean arrival rate

[2]The sources of these MBone sessions are: (1) NASA-space shuttle STS-80 coverage, video, starting time 11/14/1996, 16:14:09; (2) NASA-space shuttle STS-80 coverage, audio, starting time 12/4/1996, 10:54:49; (3) CBC Newsworld on-line test, audio, starting time 10/29/1996, 12:35:15.

TABLE II

STATISTICAL PARAMETERS FOR USER BEHAVIOR

Duration	0-199	200-499	500-4499	4500-5000
λ_i	7	5	2	1
m_i	2500	500	500	500
Characteristic	long stay	short stay		

λ_i and users' staying time follows an exponential distribution with mean value m_i. The values of λ_i and m_i are listed in Table II. The initial group size is 0. The simulated user activities consist of about 12000 join and 10900 leave events. The maximum group size is approximately 2800 and the group size at the end of simulation is about 1100.

In practice, users' accurate staying time will not always be available. To model the inaccuracy in users' *estimated staying time* (EST), we consider three classes of users. The first class of users do not report EST, the second class of users reports accurate EST, and the third class of users reports inaccurate EST. In the third class, the EST for user i is modelled as a random variable with Gaussian distribution $N(\mu_i, \sigma_i^2)$, and the mean value μ_i is the actual staying time [3]. We also assume that in the third class, the ratio of the standard deviation σ_i to the mean μ_i of the EST is constant across the users, and is denoted by $R = \sigma_i/\mu_i$. The probability that a user is in the first, second, and third classes is denoted by P_0, P_1, and $(1 - P_0 - P_1)$, respectively.

In the first experiment, we consider that a user either does not report EST or reports an accurate EST, *i.e.*, $P_1 = 1 - P_0$. By varying the value of P_0, the average join and leave time costs are shown in Fig. 8, where the average leave time increases with P_0 almost linearly. The only exception is the data point at $P_0 = 1$, *i.e.*, when no user reports EST. When $P_0 = 1$, the average join, leave, and overall time costs are: $T_{join} = 1.87$, $T_{leave} = 10.97$, and $T = 6.22$. This is the situation when the exit tree is not activated during the group lifetime. From these data, we can see that when the exit tree is not used, the average overall time cost is equal to or lower than the costs when $P_0 \geq 0.8$. This is because activating the exit tree increases the depth of the key tree by one. When P_0 is large, a large portion of users without departure information cannot take advantage of the exit tree, and the overhead of the exit tree structure outweighs its benefit. The benefit of the exit tree is substantial as long as more than 30% (corresponding to $P_0 = 0.7$) or more users report accurate EST. We have also compared the performance of JET with that of TGDH in Fig. 8, where the performances of TGDH are shown as horizontal lines because they

[3]Because of the Gaussian distribution, a user could report a negative staying time. Such a case is treated as EST unavailable.

do not vary with probability P_0. We can see that JET always outperform TGDH in terms of the overall time cost and the join time cost. For user leave time cost, JET will outperform TGDH as long as more than 35% of the users report accurate EST.

The average leave time presented in Fig. 8 consists of three parts, namely, the cost of users leaving from the exit tree, from the main tree, and from the join tree, respectively. We illustrate these three parts in Fig. 9. In particular, to obtain the first part, we obtain the average leave time for users leaving from the exit tree; then we multiply it with the percentage of user departures from the exit tree with respect to the total number of user departure events. The other two parts can be computed similarly and the average leave cost is the summation of these three parts. We can see that when P_0 is small, the user leave time is dominated by the time cost of the users leaving from the exit tree. As P_0 increases, the user leave time is gradually dominated by the time cost of users leaving from the main tree. In this experiment, most users will stay in the group for a non-trivial period of time, therefore the percentage of users leaving from the join tree is very small.

In the second experiment, we consider that all users will report EST ($P_0 = 0$). The degree of deviation is at $R \in \{0.1, 0.2, 0.3\}$, and P_1 varies in the range of [0,1]. The average join and leave time under different P_1 values are plotted in Fig. 10. We can see that when the proportion of inaccurate estimates $(1 - P_1)$ is small, the proposed JET scheme can achieve good time efficiency in both join and leave events. However, the average leave time is sensitive to the change in R value, especially in the range where $(1 - P_1)$ is small, where the gain obtained by using the exit tree diminishes quickly with the increase of R.

In the third experiment, all users report inaccurate EST, which corresponds to $P_0 = P_1 = 0$. The average join and leave time costs are simulated when the value of R is in the range of [0,1]. Fig. 11 shows that, when the standard deviation is two orders of magnitude smaller than the true staying time ($R \leq 0.01$), the proposed JET scheme can efficiently manage both user join and leave events. We also note from Fig. 10 and Fig. 11 that, when a large portion of users do not report accurate estimation, the advantage of the exit tree diminishes.

Table III lists the average and the worst case time costs for JET with both join and exit tree (when $P_0 = 0$, $P_1 = 0.1$ and $R = 0.1$), JET using only the join tree, and TGDH. All the worst case time costs do not change with the simulation parameters (P_0, P_1, and R). Comparing the performance of JET using only the join tree, which does not depend on users' reported EST, with that of JET using both join and exit tree, we can see that the exit tree indeed provides a reduction in terms of the average overall time cost, even when around 10% of users report inaccurate EST. Comparing these time costs with those of

TABLE III

SIMULATED DATA EXPERIMENT: $P_0 = 0,\ P_1 = 0.1,\ R = 0.1$

	average			worst case	
	join	leave	overall	join	leave
JET (join tree only)	1.87	10.97	6.22	13	13
JET (join and exit tree)	3.25	6.34	4.73	14	14
TGDH	10.83	9.96	10.41	12	12

TGDH, we can see that the proposed JET scheme can improve time efficiency in terms of the average time costs, while tolerating a small amount of inaccuracy in EST. However, for a group of size n, the worst case operation time of JET using only the join tree is $\lceil \log n \rceil + 1$, and that of JET using both join and exit tree is $\lceil \log n \rceil + 2$. These worst case time costs are one and two more rounds than that of TGDH, respectively. This is due to the increased depth by the join and exit tree structure, which we refer to as the *structural overhead* of the JET scheme.

Two observations can be made from the above experiments. First, regardless of the accuracy in EST, the join tree scheme can improve the time efficiency for join events. Second, although the inaccuracy in EST comes in different forms (no EST or inaccurate EST), the overall operation time is not very sensitive to the change of experiment parameters P_0, P_1, and R. This is because inaccurate EST leads to user departures from the main tree. When users leave from the main tree, we simultaneously relocate the users from the join tree to the main tree (refer to Section IV-C). As such, part of the join tree relocation cost can be amortized by the leave cost. Such an amortized cost can be counted either toward the join time or toward the leave time. In this paper, we have counted it toward the leave time. Therefore when we see a cost increase in user leave events, we will often see a cost reduction in user join events, which will partially offset the overall cost increase in the overall efficiency.

VI. DISCUSSIONS

A. Extension to Multi-Level Join Tree

The idea of caching the joining and leaving users, as in the design of memory hierarchy, can be extended to multiple levels. Here we illustrate an extension of the one-level join tree to a two-level join tree. We consider a new join tree topology, where a smaller join tree is attached directly to the tree root, a larger join tree is attached one level lower from the tree root with the main tree as its sibling sub-tree. We refer to the smaller join tree as Level-1 join tree and the larger one as Level-2 join tree, respectively.

Such a topology can be visualized as in Fig. 2(a), where the exit tree is replaced by the Level-2 join tree. We note that the exit tree does not exist in this topology.

We compare the average join time using a two-level join tree with that using a one-level join tree, and try to find the condition under which the two-level join tree has advantage over the one-level join tree. From our analysis, the smallest group size that can benefit from a two-level join tree is around or greater than 180. Compared to the activation group size of 8 for the one-level join tree, the result shows that the two-level join tree would improve the rekeying time efficiency when the group grows larger.

B. The Implementation of Key Agreement

In this subsection, we discuss two implementation methods for the proposed JET protocol with and without a group coordinator. We show that these two implementations will give the same time cost.

In the first case we consider using a *group controller* in the implementation of JET. In [11], group controller was suggested to be one of the group members who knows all group membership information and facilitates adding and excluding members. However, the group controller does not have the knowledge of the secret keys of other members and therefore would not violate the security requirements of the contributory key management. The joining/leaving user will send a request to the group controller. Then the group controller sends a broadcast message specifying the change in the logical key tree, including the insertion node ID, adjustment of the key tree structure, etc. For each join tree relocation or batch movement to the exit tree, the group controller will also send a broadcast message to specify where each user will be relocated to. The group controller would have a storage overhead proportional to the group size as it needs to store the topology of the key tree. In addition, the group controller has a communication overhead of one broadcast message per join or leave event and one broadcast message for each batch relocation, which takes place infrequently. When the current group controller leaves the group, another member in the group is chosen as the new group controller and the related information is passed from the leaving group controller to the new one. The group controller can also be a non-member entity.

The second implementation does not require a group controller. Instead, each group member stores the topology of the key tree and follows the JET protocol. Thus all users can achieve consistent action for key update. During a join event, the joining user will broadcast a request to the whole group. All members will find the same insertion node according to the insertion strategy, then a *sponsor* is chosen as the rightmost leaf node in the subtree rooted at the insertion node [12]. Since the sponsor knows all the subgroup keys along the path from the insertion node to the root, the sponsor will perform a two-party DH key exchange with the new user, compute the keys on the path from the insertion node to the tree

TABLE IV

A COMPARISON OF REKEYING PROTOCOL COMPLEXITY (GROUP SIZE n)

	Key update time in group lifetime		Rekeying message overhead (# of messages)		Computation overhead (# of exponentiations)	
	for one user	for entire system	with multicast	without multicast	for one user	for all users
JET	$O(n)$	$O(n \log(\log n))$	$O(\log n)$	$O(n)$	$O(\log(\log n))$	$O(n)$
TGDH	$O(n \log n)$	$O(n \log n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$

root, and then broadcast the blinded keys on this path. The blinded keys are the results of exponentiation base g raised to the power of the secret keys, which can be public known. Such a procedure is detailed in [12]. When a leave event occurs, the sponsor is chosen as the rightmost leaf node of the subtree rooted at the leaving user's sibling node, and a key update procedure similar to that of a join event takes place. Since all users have the same view of the key tree structure, they can also cooperate in the update operations of the key tree, such as the batch relocation/movement.

An important factor in the rekeying cost is the depth of the joining/leaving node, d. In the first implementation, d rounds of DH key exchange will be performed in key update, which has $2d$ exponentiations and d message transmissions in total. in the second implementation, the sponsor needs to compute d un-blinded keys, d blinded keys, and send d messages. Therefore, the two implementations have the same total computation and communication cost, as well as the same time complexity. But in the second implementation, half of the computation load and almost all communication load are on the sponsor. As for the storage overhead, in the first implementation, only the group controller needs to store the structure of the key tree, whose size is proportional to the group size. In the second implementation, each user needs to store a copy of the key tree structure and the total storage overhead grows proportional to the square of the group size.

C. Protocol Complexity

In this subsection, we provide a more comprehensive comparison of the protocol complexity between JET and TGDH. The complexity aspects we consider are the rekeying time cost during a group lifetime, the messaging overhead, and the computation overhead. These results are summarized in Table IV.

1) Time Complexity from Other Perspectives: Our previous discussions have been focused on the time cost from individual user's perspective and on a per event basis. Two additional aspects can help evaluate the overall efficiency of key management from a system perspective. One is the amount of time a user spends on key update during his/her lifetime in the group, and the other is the amount of time the whole group spends on key update during the lifetime of the group.

Consider a sequence of n join events followed by n leave events. We assume that the first user joining the group is also the last one to leave the group. In JET, such a user will spend most of the life time in the main tree for key management purpose. On average, this user will spend 2-round time for key update with each user join event and 3-round time with each user leave event, assuming all users report their staying time accurately. Therefore this user has spent $\Theta(n)$ rounds in total on key update during the life time. Since this user has the longest life time among all users, $O(n)$ is the upper bound for any user's total key update time. For tree-based key agreement using a simple key tree, this first-come-last-leave user will spend $O(n \log n)$ rounds in total on key update.

If we consider a group of n users as a whole, for the same sequence of events described above, the group will spend $O(n \log(\log n))$ rounds in key update using JET. If a key agreement using a simple key tree is employed, the time cost will be $O(n \log n)$. Compared to TGDH, we note that the improvement from a system perspective is not as significant as that from a user's perspective.

2) Communication Complexity: During member join and leave, a joining/leaving member should send a join/leave request. Afterwards, in the rekeying process, at least two messages will be sent for each DH key exchange. This messaging overhead is the communication cost of the rekeying protocol.

We now discuss the average number of messages for user join and leave events in JET protocol. In the first scenario, we consider that multicast is available. In particular, if a message needs to be sent to m users, sending one multicast message would suffice. In this case, the average number of messages is $O(\log n)$ for both join and leave events. In the second scenario, we consider that multicast is not available. If a message needs to be sent to m users, m duplicate copies of the same message must be sent. In this case the average number of messages is $O(n)$ for both user join and leave event. From Table IV we can see that, the rekeying message overhead in JET is comparable to those in TGDH.

3) Computation Complexity: In the proposed JET protocol, the total number of exponentiations performed by all users is $O(n)$ during the key update for a join or leave event. Such a measurement captures the overall computation load of the entire group.

During each join or leave event, the number of exponentiations performed by any individual member is less than or equal to two times the number of DH rounds. Therefore for any single user, the average number of exponentiations is also $O(\log(\log n))$ per join/departure event.

VII. Conclusions

In this paper, we have presented a new contributory key agreement, known as the Join-Exit-Tree Group Key Agreement, for secure group communications. Built upon tree-based group key management, the proposed scheme employs a main tree as well as join and exit subtrees that serve as temporary buffers for joining and leaving users. To achieve time efficiency, we have shown that the optimal subtree capacity is at the log scale of the group size and designed an adaptive algorithm to activate and update join and exit subtrees. As a result, the proposed JET scheme can achieve an average time cost of $O(\log(\log n))$ for user join and leave events in a group of n users, and reduces the total time cost of key update over a system's life time from $O(n \log n)$ by prior works to $O(n \log(\log n))$. In the meantime, the proposed scheme also achieves low communication and computation overhead. Our experimental results on both simulated user activities and the real MBone data have shown that the proposed scheme outperforms the existing tree-based schemes by a large margin in the events of group key establishment, user join, and user departure for large and dynamic groups, without sacrificing the time efficiency for small groups.

Appendix I

In this appendix, we prove the inequality (5) in Section III:

$$\frac{1}{A}\sum_{k=1}^{A} r(k) \leq \frac{1}{2}\log A + 1, \tag{20}$$

where $r(1) = 1$, $r(2^p + q) = 1 + r(q)$, p is a non-negative integer, and $q \in [1, 2^p]$ is a positive integer. The equality holds when A is a power of 2.

We first use induction to show that when $A = 2^p$, $p = 0, 1, 2, ...$, the equality holds.

When $A = 1$, LHS = RHS = 1.

Next, we assume the equality holds for $A = 2^p$, namely,

$$\frac{1}{2^p}\sum_{k=1}^{2^p} r(k) = \frac{1}{2}\log 2^p + 1. \tag{21}$$

Consider the case of $A = 2^{p+1}$.

$$\begin{aligned} LHS &= \frac{1}{2^{p+1}}\sum_{k=1}^{2^{p+1}} r(k) \\ &= \frac{1}{2^{p+1}}\left(\sum_{k=1}^{2^p} r(k) + \sum_{k=1}^{2^p}(r(k)+1)\right) \\ &= \frac{1}{2^{p+1}}\left(2\cdot(\frac{1}{2}\log 2^p + 1)2^p + 2^p\right) \\ &= \frac{1}{2}\log 2^{p+1} + 1 = RHS, \end{aligned} \tag{22}$$

where (22) is obtained using the induction assumption (21).

We now prove the inequality for any positive integer A. It is obvious to see that inequality is true for $A = 1, 2$. By induction, suppose that the inequality is true for all $1 \le A < 2^p + q$, and we consider $A = 2^p + q$, where $0 < q \le 2^p$.

$$\begin{aligned} LHS &= \frac{1}{A}\sum_{k=1}^{A} r(k) \\ &= \frac{1}{A}\left(\sum_{k=1}^{2^p} r(k) + \sum_{k=1}^{q}(r(k)+1)\right) \\ &\le \frac{1}{A}\left((\frac{1}{2}\log 2^p + 1)2^p + q(\frac{1}{2}\log q + 1) + q\right) \qquad (23) \\ &= \frac{1}{2}\left\{\frac{1}{A}(2^p \log 2^p + q\log q + 2q)\right\} + 1, \qquad (24) \end{aligned}$$

where (23) is obtained using the induction assumption.

To prove that (24) $\le \frac{1}{2}\log A + 1$ is equivalent to prove

$$\frac{2^p}{A}\log 2^p + \frac{q}{A}\log(4q) \le \log A. \qquad (25)$$

Applying the identity $\log k = \log e \cdot \ln k$ and $\ln k = \int_1^k \frac{1}{x}dx$, (25) can be written as an integration form

$$\begin{aligned} &\log e\left\{\frac{2^p}{A}\int_1^{2^p}\frac{1}{x}dx + \frac{q}{A}\int_1^{4q}\frac{1}{x}dx\right\} \le \log e\int_1^A \frac{1}{x}dx \\ &\Leftrightarrow 2^p\int_{2^p}^{A}\frac{1}{x}dx + q\left[\int_1^A \frac{1}{x}dx - \int_1^{4q}\frac{1}{x}dx\right] \ge 0 \qquad (26) \end{aligned}$$

We denote $B = 2^p$ and fix p (hence B is fixed). Thus $A = B + q$. It is straightforward to see that (26) holds when $B + q \ge 4q$, or $1 \le q \le \frac{B}{3}$.

When $B/3 \le q \le B$, (26) is equivalent to

$$\frac{2^p}{A}\int_{2^p}^{A}\frac{1}{x}dx - \frac{q}{A}\int_A^{4q}\frac{1}{x}dx \ge 0. \qquad (27)$$

Since q is the only variable in (27), let $f(q)$ be the LHS of (27), and consider $f(q)$ as a continuous function of q

$$f(q) = \frac{B}{B+q}\int_B^{B+q}\frac{1}{x}dx - \frac{q}{B+q}\int_{B+q}^{4q}\frac{1}{x}dx,$$

where $q \in [B/3, B]$. Taking the derivative of $f(q)$, we get

$$\frac{d}{dq}f(q) = -\frac{B}{(B+q)^2}\int_B^{4q}\frac{1}{x}dx < 0. \qquad (28)$$

In previous proof we showed that the equality of (20) holds when A is power of 2, i.e. $f(B) = 0$. We also showed that $f(q) > 0$ for $1 \le q \le \frac{B}{3}$. Since $f(B/3) > 0$, $f(B) = 0$, $f(q)$ is continuous on $[B/3, B]$ and $f'(q) < 0$, we must have $f(q) > 0$ on $[B/3, B]$. Thus (26) also holds for $B/3 \le q \le B$. This completes the proof.

Appendix II

In this appendix, we prove the following inequality in Section III-A.4: for any $x > 8$,

$$\log x > \frac{2 \ln x}{2 \ln x - 1} \log(2 \ln x). \tag{29}$$

Let $y = \ln x$. We consider the case when the group size is larger than 1, so $x \in (1, +\infty)$, and $y \in (0, +\infty)$. Under such a condition, (29) becomes

$$2y - 1 > 2 \ln 2(1 + \log y). \tag{30}$$

Let $g(y) = (2y - 1) - 2 \ln 2(1 + \log y)$. Function $g(y)$ havs two zeros at $y_0 = 1/2$ and $y_1 \approx 1.7564$. In addition, $g(y) > 0$ for any $y > y_1$. Therefore (29) holds for any $x > e^{y_1} \approx 5.7917$. In our proposed protocol, we choose a larger threshold value 8 as eight users lead to a balanced main tree.

References

[1] M. J. Moyer, J. R. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," *IEEE Network*, pp. 12–23, Nov./Dec. 1999.

[2] S. Paul, *Multicast on the Internet and its applications*, Kluwer academic Publishers, 1998.

[3] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and Communications Security*. 2002, pp. 41–47, ACM Press.

[4] P. Judge and M. Ammar, "Gothic: A group access control architecture for secure multicast and anycast," in *Proceedings of the IEEE INFOCOM'02*, 2002, pp. 1547–1556.

[5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," in *Proceedings of the IEEE INFOCOM'99*, 1999, pp. 708–716.

[6] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, Feb 2000.

[7] A. Perrig, D. Song, and J. D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2001, pp. 247–262.

[8] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) specification," RFC 2093, July 1997.

[9] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architecture," Internet-Draft draft-wallner-key-arch-00.txt, June 1997.

[10] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key management for secure internet multicast using boolean function minimization techniques," in *Proceedings of the IEEE INFOCOM'99*, 1999, vol. 2, pp. 689–698.

[11] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: a new approach to group key agreement," in *Proceedings of the 18th International Conference on Distributed Computing Systems*, 1998, pp. 380–387.

[12] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*. 2000, pp. 235–244, ACM Press.

[13] L. R. Dondeti and S. Mukherjee, "DISEC: a distributed framework for scalable secure many-to-many communication," in *Proceedings of the 5th IEEE Symposium on Computers and Communications*, 2000, pp. 693–698.

[14] S. E. Eldridge and C. D. Walter, "Hardware implementation of montgomery's modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 693–699, June 1993.

[15] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) architecture," RFC 2094, July 1997.

[16] Y. Sun, W. Trappe, and K. J. R. Liu, "A scalable multicast key management scheme for heterogeneous wireless networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 4, pp. 653–666.

[17] R. Molva and A. Pannetrat, "Scalable multicast security in dynamic groups," in *Proceedings of the 6th ACM conference on Computer and Communications Security*, 1999, pp. 101–112.

[18] S. Mittra, "Iolus: a framework for scalable secure multicasting," in *Proceedings of the ACM SIGCOMM'97*. 1997, pp. 277–288, ACM Press.

[19] S. Banerjee and B. Bhattacharjee, "Scalable secure group communication over IP multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1511–1527, Oct. 2002.

[20] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersayKey framework: Versatile group key management," *IEEE Journal on Selected Areas in Communications*, pp. 1614–1631, Sept. 1999.

[21] W. Trappe, Y. Wang, and K. J. R. Liu, "Resource-aware conference key establishment for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 134–146, Feb. 2005.

[22] B. Sun, W. Trappe, Y. Sun, and K. J .R. Liu, "A time-efficient contributory key agreeement scheme for secure group communications," in *Proceedings of the IEEE International Conference on Communications*, 2002, pp. 1159–1163.

[23] S. Zhu, S. Setia, and S. Jajodia, "Performance optimizations for group key management schemes," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003, pp. 163–171.

[24] Y. Mao, Y. Sun, M. Wu, and K. J. R. Liu, "Dynamic join-exit amortization and scheduling for time-efficient group key agreement," in *Proceedings of the IEEE INFOCOM 2004*, vol. 4, pp. 2617 – 2627.

[25] I. Ingemarsson, D. T. Tang, and C. K. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. IT-28, no. 5, pp. 714–720, September 1982.

[26] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," in *Proceedings of the 3rd ACM conference on Computer and Communications Security*. 1996, pp. 31–37, ACM Press.

[27] K. Becker and U. Wille, "Communication complexity of group key distribution," in *Proceedings of the 5th ACM conference on Computer and Communications Security*. 1998, pp. 1–6, ACM Press.

[28] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Proceedings of the EUROCRYPT'94*. 1994, pp. 275–286, LCNS 950.

[29] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, November 1976.

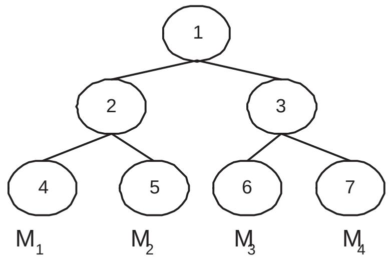
[30] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann publishers, second edition, 1996.

[31] T. H. Corman, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press and McGraw-Hill Book Company, second edition, 2001.

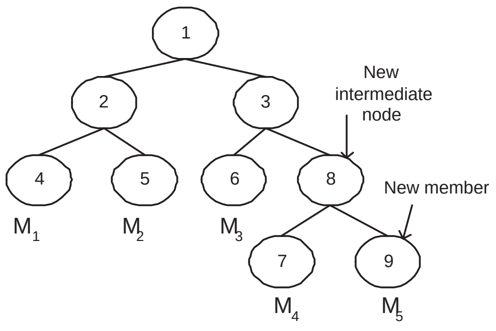
[32] MBone user activity data, "ftp://ftp.cc.gatech.edu/people/kevin/release-data," March 2003.

[33] K. C. Almeroth and M. H. Ammar, "Multicast group behavior in the Internet's multicast backbone (MBone)," *IEEE Communications Magazine*, pp. 124–129, June 1997.

[34] K. C. Almeroth, "A long-term analysis of growth and usage patterns in the multicast backbone (MBone)," in *Proceedings of the IEEE INFOCOM'00*, March 2000, vol. 2, pp. 824–833.

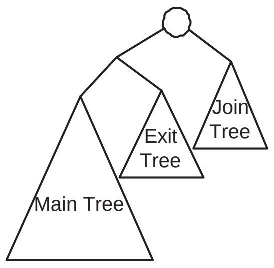


(a) a key tree

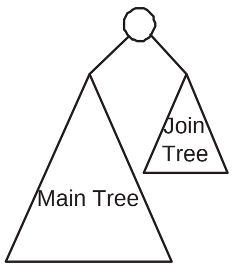


(b) user join

Fig. 1. Notations for a key tree.



(a) join, exit, and main tree



(b) join and main tree

Fig. 2. Topology for the proposed join-exit tree.

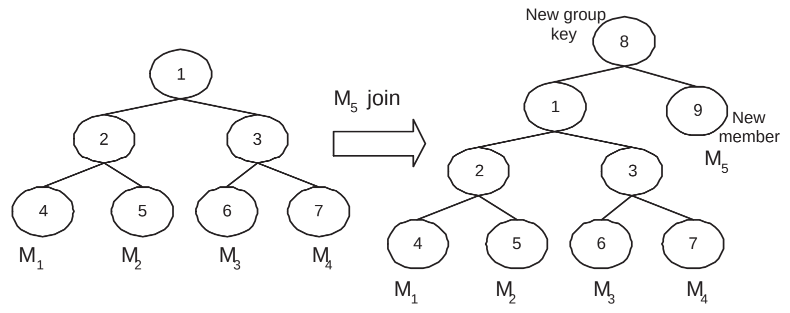


Fig. 3. User join at the join tree root. Note that the new user M_5 becomes the root of the join tree.

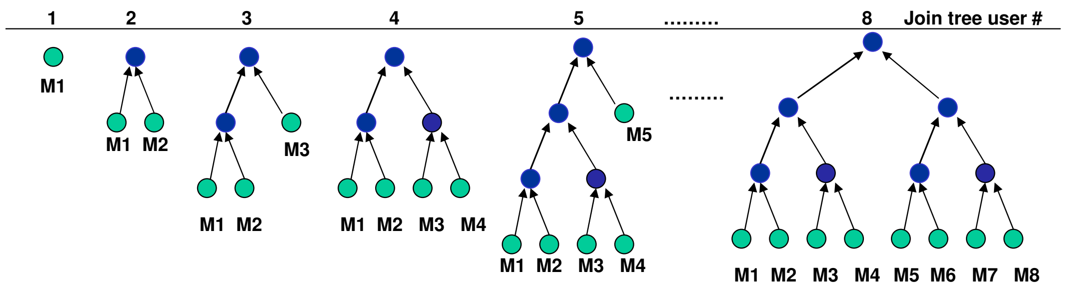
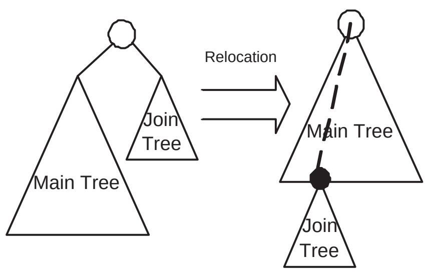
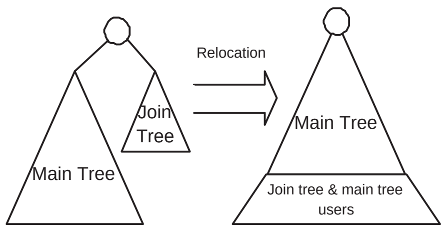


Fig. 4. Sequential user join strategy (only the join tree is shown).



(a) Method 1



(b) Method 2

Fig. 5. Relocation methods for the join tree.

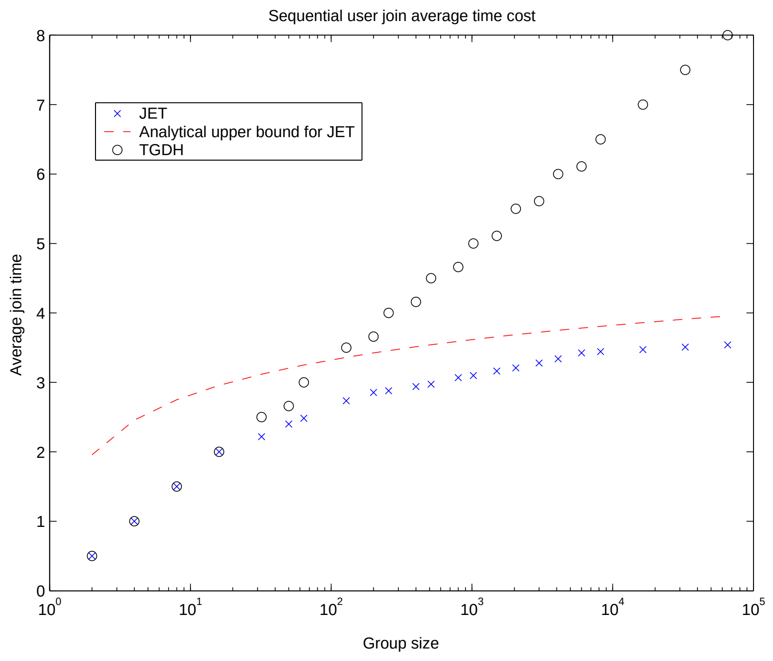


Fig. 6. Average time cost for sequential user join.

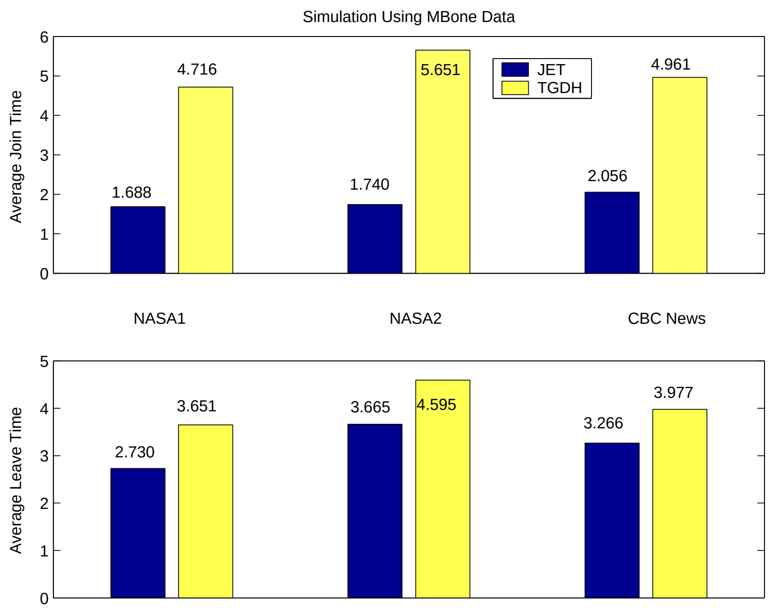


Fig. 7. Average join and leave time for simulations using MBone data.

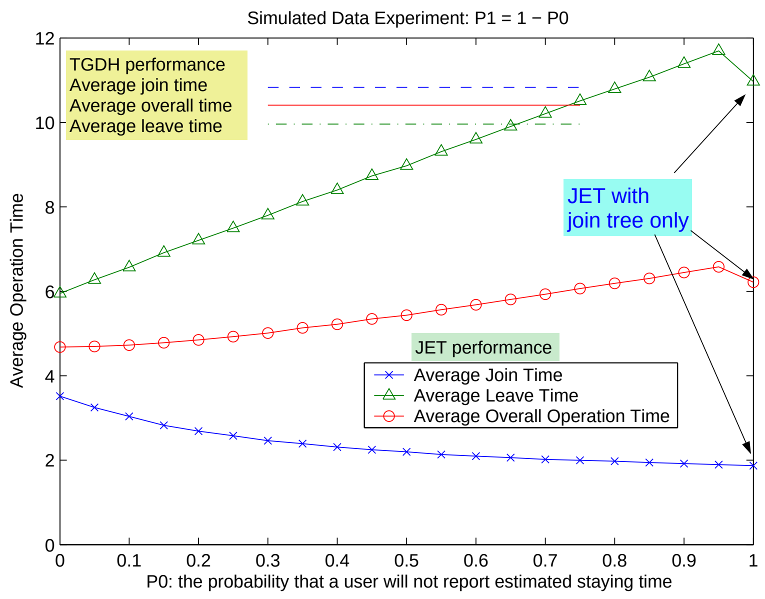


Fig. 8. Average join, leave and overall time costs for the first experiment using simulated data. A user either do not report EST with probability P_0, or reports accurate EST with probability $P_1 = 1 - P_0$.

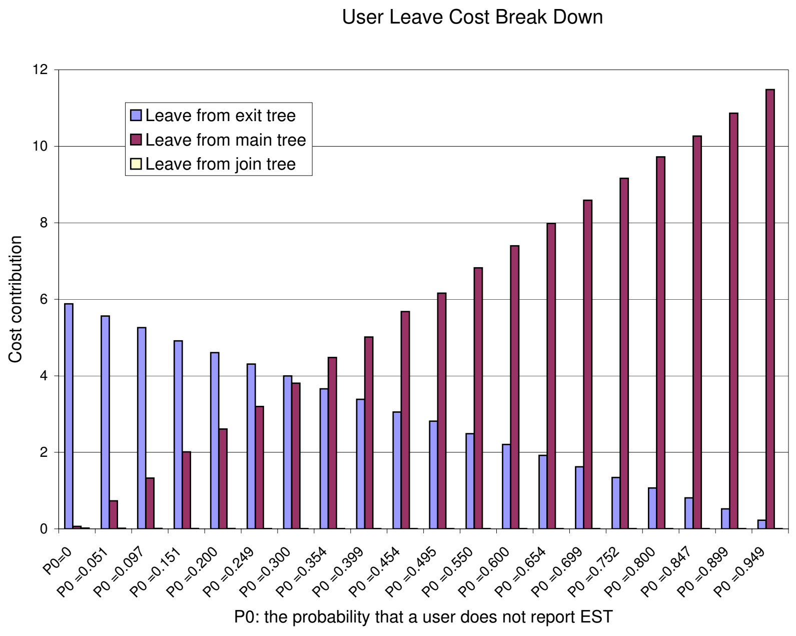


Fig. 9. A breakdown of the contributions to the user leave time in Fig. 8. Shown in the figure is the contribution to the user leave time by users leaving from the exit tree, the main tree, and the join tree. These contributions (in rounds) is plotted against P_0, with $P_1 = 1 - P_0$.

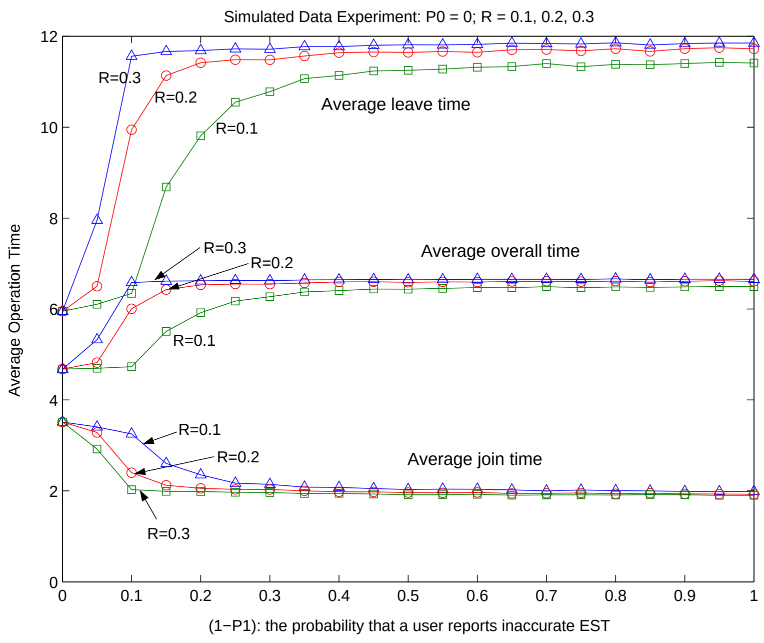


Fig. 10. Average join, leave and overall time costs for the second experiment using simulated data. A user either reports accurate EST with probability P_1, or reports inaccurate EST with probability $1-P_1$. For inaccurate EST, the deviation parameter $R \in \{0.1, 0.2, 0.3\}$.

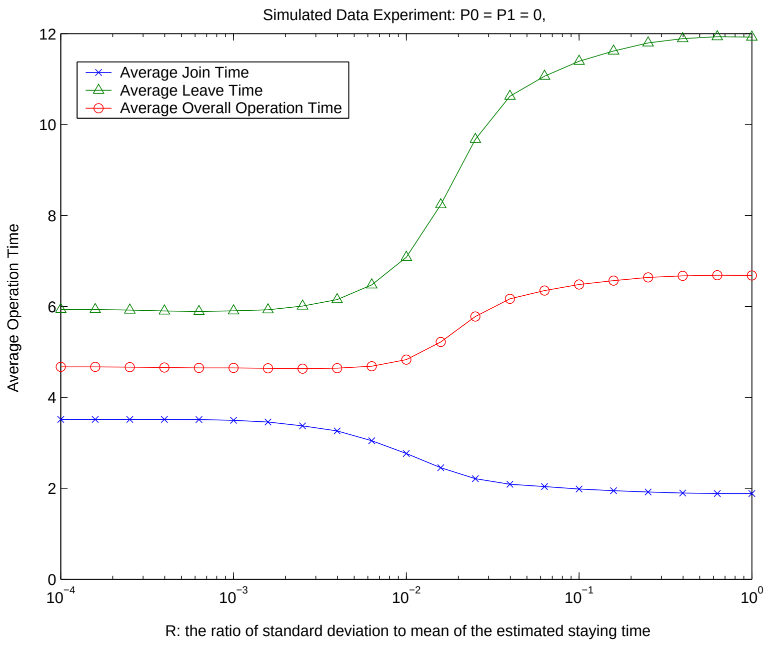


Fig. 11. Average join, leave and overall time costs for the third experiment using simulated data. All users report inaccurate EST. The deviation parameter $R \in [10^{-4}, 1]$.